Sage 퀵 레퍼런스: 선형대수학

Robert A. Beezer, 박진영, 이상구

Sage Version 4.1

http://wiki.sagemath.org/quickref

Based on work by Peter Jipsen, William Stein

한글화: 성균관대학교 수학과

벡터 생성

주의: 인덱스는 0부터 시작

`u = vector(QQ, [1, 3/2, -1])` 원소가 세 개인 유리수 벡터

`v = vector(QQ, {2:4, 95:4, 210:0})`

크기가 210인 유리수 벡터. 3번째와 96번째 원소가 4이다.

벡터 연산

```
u = vector(QQ, [1, 3/2, -1])
v = vector(ZZ, [1, 8, -2])
```

`2*u - 3*v` 일차 결합

`u.dot_product(v)` u, v 내적

`u.cross_product(v)` u, v 외적

`u.inner_product(v)` u, v 내적

`u.pairwise_product(v)` 인덱스에 맞춰서 곱

`u.norm() == u.norm(2)` 유클리드 norm $\mathbf{u} = \sqrt{\sum_i u_i^2}$

`u.norm(1)` 원소들의 합 $||\mathbf{u}||_1 = \sum_i u_i$

`u.norm(Infinity)` 최대 원소값 $||\mathbf{u}||_\infty = \max(u_i)$

행렬 생성

주의: 행, 열의 인덱스는 0부터 시작

`A = matrix(ZZ, [[1,2],[3,4],[5,6]])`

3×2 정수 행렬

`B = matrix(QQ, 2, [1,2,3,4,5,6])`

2×3 유리수 행렬

`C = matrix(CDF, 2, 2, [[5*I, 4*I], [I, 6]])`

53-bit precision을 가지는 2×2 복소수 행렬

`Z = matrix(QQ, 2, 2, 0)` 영행렬

`D = matrix(QQ, 2, 2, 8)`

대각원소가 모두 8인 2×2 대각행렬

`I = identity_matrix(5)` 5×5 단위행렬

`J = jordan_block(-2,3)`

3×3 행렬, -2 on diagonal, 1's on super-diagonal

`var('x y z'); K = matrix(SR, [[x,y+z],[0,x^2*z]])`

x, y, z를 미지수로 가지는 2×2 symbolic 행렬

`L=matrix(ZZ, 20, 80, {(5,9):30, (15,77):-6})`

20×80, 5행 9열은 30, 15행 77열은 -6이고 나머지는 모두 0인 정수행렬

행렬 곱

`u = vector(QQ, [1,2,3])`, `v = vector(QQ, [1,2])`

```
A = matrix(QQ, [[1,2,3],[4,5,6]])
B = matrix(QQ, [[1,2],[3,4]])
```

`u*A`, `A*v`, `B*A`, `B^6`, `B^(-3)` 모두 가능

`B.iterates(v, 6)` $== vB^0, vB^1, \ldots, vB^5$

`rows = False` moves `v` to right of matrix powers

`f(x)=x^2+5*x+3` 이고 `f(B)` 도 가능

`B.exp()` 행렬 exponential, i.e. $\sum_{k=0}^{\infty} \frac{B^k}{k!}$

행렬 공간

`M = MatrixSpace(QQ, 3, 4)`

차원이 12인 3×4 행렬 공간

`A = M([1,2,3,4,5,6,7,8,9,10,11,12])`

`M`의 원소인 3×4 행렬

`M.basis()` 기저

`M.dimension()` 차원

`M.zero_matrix()` `M`의 원소인 3×4 영행렬

행렬 연산

`5*A+2*B` 일차결합

`A.inverse()`, `A^(-1)`, `~A`

`A`의 역행렬, 만약 `A`가 singular면 `ZeroDivisionError` 발생

`A.transpose()` 전치행렬

`A.antitranspose()` 전치행렬에 역순서 정렬을 취한 행렬

`A.adjoint()` 수반행렬

`A.conjugate()` 켤레행렬

`A.restrict(V)` 벡터공간 V로 제한을 둔 행렬

행 연산

Row Operations: (change matrix in place)

주의: 행의 인덱스는 0부터 시작

`A.rescale_row(i,a)` `a*`(row `i`)

`A.add_multiple_of_row(i,j,a)` `a*`(row `j`) + row `i`

`A.swap_rows(i,j)`

열 연산도 가능 `row→col`

새로운 행렬을 만들자면, `B = A.with_rescaled_row(i,a)`

Echelon Form

`A.echelon_form()`, `A.echelonize()`, `A.hermite_form()`

주의: ring에 따라 결과가 다름

```
A = matrix(ZZ,[[4,2,1],[6,3,2]])
B = matrix(QQ,[[4,2,1],[6,3,2]])
```

`A.echelon_form()` `B.echelon_form()`

$$\begin{pmatrix} 2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} 1 & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

`A.pivots()` 열공간에서 대표인자들을 선택

`A.pivot_rows()` 행공간에서 대표인자들을 선택

부분 행렬

주의: 행, 열의 인덱스는 0 부터 시작

`A = matrix(ZZ,[[4,2,1],[6,3,2]])`

`A.nrows()` 행 개수

`A.ncols()` 열 개수

`A[i,j]` 행`i` 열`j`의 원소

주의: 가능: `A[2,3] = 8`, 불가능: `A[2][3] = 8`

`A[i]` Python 튜플(tuple) 형식의 행 `i`

`A.row(i)` Sage 벡터(vector) 형식의 행 `i`

`A.column(j)` Sage 벡터(vector) 형식의 행 `j`

`A.list()` 행 우선 순서로 하나의 Python 리스트(list)

`A.matrix_from_columns([0,1,0])`

`A`의 0, 1, 0번째 열벡터를 순서대로 하여 새로운 행렬 생성

`A.matrix_from_rows([1, 0, 0])`

`A`의 1, 0, 0번째 행벡터를 순서대로 하여 새로운 행렬 생성

`A.matrix_from_rows_and_columns([0,1,0],[1,0])`

`A`의 0, 1, 0번째 행벡터의 1번, 0번째 원소로 새로운 행렬 생성

`A.rows()` 모든 행벡터를 tuple로 한 리스트

`A.columns()` 모든 열벡터를 tuple로 한 리스트

`A.submatrix(i,j,nr,nc)`

행과 열 `(i,j)`에서 시작하여 `nr` 행과 `nc` 열만큼으로 새로운 행렬 생성

`A[0:1,1:2]`, `A[0:2:2,1::-1]` Python 스타일 리스트 슬라이싱

행렬 결합

`A.augment(B)` `A` 행렬 오른쪽에 `B` 행렬을 붙인 첨가행렬

`A.stack(B)` `A` 행렬 아래에 `B` 행렬을 붙인 첨가행렬

`A.block_sum(B)` `A` 행렬을 왼쪽 위에, `B` 오른쪽 아래에 넣은 대각행렬

`A.tensor_product(B)` Multiples of `B`, arranged as in `A`

스칼라 값을 내놓은 행렬

`A.rank()` 행렬 `A`의 rank

`A.nullity() == A.left_nullity()` 행렬 `A`의 nullity. Sage에서는 left nullity와 동일.

`A.right_nullity()` 행렬 `A`의 right nullity. 현대 선형대수학 교재에서 nullity는 이것을 얘기함.

`A.determinant() == A.det()` 행렬 `A`의 행렬식

`A.permanent()`

`A.trace()` 행렬 `A`의 대각합

`A.norm() == A.norm(2)` 유클리드 norm

`A.norm(1)` 열벡터 원소들 합 중 가장 큰 것.

`A.norm(Infinity)` 행벡터 원소들 합 중 가장 큰 것.

`A.norm('frob')` Frobenius norm

행렬 속성

`.is_zero()` (영행렬?), `.is_one()` (단위행렬?),

`.is_scalar()` (단위행렬의 스칼라곱?),

`.is_square()` (정사각행렬?),

`.is_symmetric()` (대각행렬?),

`.is_invertible()` (가역행렬?),

`.is_nilpotent()` (Nilpotent 행렬?)

고유값

`A.charpoly('t')` 특성방정식, (t를 주지 않으면 x에 대한 방정식을 구함

`A.characteristic_polynomial() == A.charpoly()`

`A.fcp('t')` 인수분해된 특성방정식

`A.minpoly()` 최소다항식(minimal polynomial)

`A.minimal_polynomial() == A.minpoly()`

`A.eigenvalues()` (중복도를 포함, 순서에 관계없음)

`A.eigenvectors_right()` 고유벡터를 구함

고유값 한 값에 대하여 다음을 구함:

`e`: 고유값

`V`: 고유공간의 기저; 고유벡터

`n`: 대수적 중복도

`A.eigenmatrix_right()` 고유벡터로 구성된 행렬을 구함 (행렬의 닮음)

두 행렬값을 구함:

`D`: 고유값을 대각성분으로 가지는 대각행렬

`P`: 고유벡터를 열벡터로 가지는 행렬

만일 행렬이 대각화가능하지 않으면, `P`의 모든 열벡터는 영벡터임

분해

Note: 행렬의 기본을 두는 공간에 따라 결과가 약간 다를 수 있음

`A.jordan_form(transformation=True)`

다음 두 행렬을 구함:

`J`: 행렬의 고유값에 따른 Jordan blocks

`P`: 가역행렬

so `A == P^(-1)*J*P`

`A.smith_form()` 다음 행렬들을 구함:

`D`: 대각성분에 elementary divisors를 포함한 행렬

`U, V`: unit determinant

so `D == U*A*V`

`A.LU()` 다음 행렬들을 구함:

`P`: 치환행렬(a permutation matrix)

`L`: 하삼각행렬(lower triangular matrix)

`U`: 상삼각행렬(upper triangular matrix)

so `P*A == L*U`

`A.QR()` 다음 두 행렬을 구함:

`Q`: 직교행렬(an orthogonal matrix)

`R`: 상삼각행렬(upper triangular matrix)

so `A == Q*R`

`A.SVD()` 다음 행렬들을 구함:

`U`: 직교행렬(an orthogonal matrix)

`S`: 특이값을 대각성분으로 가지는 행렬

`V`: 직교행렬(an orthogonal matrix)

so `A == U*S*(V-conjugate-transpose)`

`A.symplectic_form()`

`A.hessenberg_form()`

`A.cholesky()`

선형연립방정식의 해법

`A.solve_right(B)` `_left` too

`A*X = B`, 단 `X`는 벡터 **또는** 행렬도 가능

`A = matrix(QQ, [[1,2],[3,4]])`
`b = vector(QQ, [3,4])`
이라면 `A\b`의 답으로 `(-2, 5/2)`를 구한다.

벡터 공간

`U = VectorSpace(QQ, 4)` 4차원 유리수 공간
`V = VectorSpace(RR, 4)` 4차원 실수공간 (53-bit precision)
`W = VectorSpace(RealField(200), 4)`
4차원 실수공간(200 bit precision)
`X = CC^4` 4차원 복소수공간
`Y = VectorSpace(GF(7), 4)` 4차원의 GF(7)의 유한차원 공간
`Y.finite()`으로 확인하면 `True`
임을 알 수 있음 `len(Y.list())`를 통하여 총 $7^4 = 2401$ 개의 원소가 공간안에 존재

벡터 공간 속성

`V.dimension()`
`V.basis()`
`V.echelonized_basis()`
`V.has_user_basis()` 다른 기저를 가지는지 확인?
`V.is_subspace(W)` 만약 `W`가 `V`의 부분공간이라면 `True`
`V.is_full()` 기저의 개수가 차원과 같은지 확인(Module에서 사용)?
`Y = GF(7)^4`, `T = Y.subspaces(2)`
`T`는 `Y`의 2차원 부분공간을 생성

부분공간 생성

`span([v1,v2,v3], QQ)`는 세개의 벡터에 의해 생성된 공간을 의미

만약 `V`와 `W`가 부분공간이라면,
`V.quotient(W)` `W`의 부분공간으로서 `V` 의 quotient를 구함
`V.intersection(W)` `V`과 `W`의 교집합
`V.direct_sum(W)` `V`과 `W`의 direct sum
`V.subspace([v1,v2,v3])` [v1,v2,v3] 벡터들에 의해 생성되는 부분공간

Dense와 Sparse

Note: 알고리즘이 일반적으로 알려진 것과 약간 다를 수 있다.
벡터와 행렬이 다음의 두가지에 따라 다르게 파악될 수 있다.
Dense: 리스트(list)를 이용한 처리
Sparse: 파이슨(Python)의 사전배열순서
`.is_dense()`, `.is_sparse()`를 통해 체크할 수 있다.
`A.sparse_matrix()` `A`의 sparse 행렬을 구한다.
`A.dense_rows()` `A`의 행벡터를 이용하여 dense한 행렬을 구한다.
`sparse` 라는 키워드를 사용할 수 있다.

환(Rings)

Note: 많은 알고리즘이 환(ring)을 기반으로 구현되어 있다.
`<object>.base_ring(R)`: 벡터와 행렬을 해당하는 환(ring)을 기반으로 만든다.
`<object>.change_ring(R)`: 벡터와 행렬을 해당하는 환(ring)을 기반으로 바꾼다.
`R.is_ring()`, `R.is_field()`
`R.is_integral_domain()`, `R.is_exact()`

대표적으로 사용하는 환(ring)과 체(field)
`ZZ` 정수, 환(ring)
`QQ` 유리수, 체(field)
`QQbar` 확장된 체(algebraic extension field)
`RDF` 실수, 체(field, inexact)
`RR` 실수, 체(53-bit, inexact)
`RealField(400)` 실수, 체(400-bit, inexact)
`CDF`, `CC`, `ComplexField(400)` 복소수, 체(field)
`RIF` 실수, 구간체(interval field)
`GF(2)` GF(2) (mod 2, 체, 특화된 구성)
`GF(p) == FiniteField(p)` `p` 소수 `p`로 구성된 체(field)
`Integers(6)` mod 6로 만들어진 환(ring)
`QuadraticField(-5, 'x')` rationals adjoin `x`=$\sqrt{-5}$
`SR` symbolic expressions으로 만들어진 환(ring)

도움말이 필요한 경우

`<command>?` 명령에 대한 요약 및 예제
`<command>??` 명령에 대한 코드소스